VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

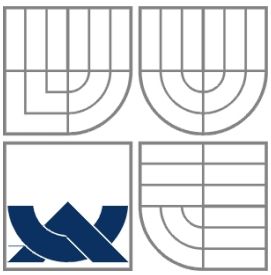
METODY PRO KLASIFIKACI DAT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

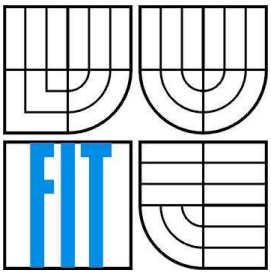
AUTOR PRÁCE
AUTHOR

PAVOL KAŠČÁK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

METODY PRO KLASIFIKACI DAT

DATA CLASSIFICATION METHODS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVOL KAŠČÁK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. VLADIMÍR BARTÍK, PH.D.

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá klasifikací dat se zaměřením na implementaci naivní Bayesovské klasifikační metody. Nejdříve je obecně popsán proces klasifikace dat, jeho rozdělení do fází s jejich charakteristikou. Následuje přesnější popis metody naivní Bayesovské klasifikace a popis implementace této metody, za použití programovacího jazyka Java a databáze MySQL. Poslední část obsahuje shrnutí dosažených výsledků.

Abstract

This bachelor's thesis deals with data classification, focusing on the implementation of naive Bayes classification method. At First, it is generally described process of data classification, its division into phases with their characteristic. It is followed by a more accurate description of the naive Bayes classification method and description of the implementation by using Java programming language and MySQL database. The last section contains a summary of the results.

Klíčová slova

Získávání znalostí z databází, dolování dat, klasifikace, klasifikační metody, naivní Bayesovská klasifikace.

Keywords

Knowledge discovery in databases, data mining, classification, classification methods, naive Bayes classification.

Citace

Pavol Kaščák: Metody pro klasifikaci dat, bakalářská práce, Brno, FIT VUT v Brně, 2010

Metody pro klasifikaci dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Pavol Kaščák
19. 5. 2010

Poděkování

Chcel by som poďakovať môjmu vedúcemu, pánovi Ing. Vladimíru Bartíkovi, Ph.D. za jeho pomoc a rady, pri tvorbe tejto práce. Ďalej svojej rodine za veľkú podporu.

Obsah

1 Úvod

Táto práca sa zaoberá klasifikáciou, ako jedným z typov úloh dolovania z dát používaných pri získavaní znalostí z databáz. Všeobecný problém pri klasifikácii je vhodný výber metódy resp. klasifikačného algoritmu. Znalosť ich vlastností nám napomáha rozhodnúť sa správne pre danú situáciu a okolnosti nasadenia metódy. Ďalej sú v práci vysvetlené základné pojmy ako, získavanie znalostí z databáz, dolovanie dát, funkcionalita dolovania z dát, obsahuje popis jednotlivých procesov, s ktorých sa získavanie znalostí skladá. Následne sa presnejšie zameriava na špecifický a zároveň najskúmanejší krok procesu získavania znalostí z databáz, ktorým je samotné dolovanie dát. Pri dolovaní dát sa uplatňujú rôzne metódy, algoritmy a postupy v závislosti na požadovanej informácii, ktorú chceme získať. Práca detailne nerieši problém jednotlivých metód, ale slúži ako demonštrácia použitia naivnej Bayesovskej klasifikácie, ako zvolenej metódy. Avšak jednotlivé metódy uplatňované pri dolovaní dát sú stručne vysvetlené a charakterizované. Z názvu by mohol plynúť dojem, že sa jedná len o akúsi prostú metódu, to samozrejme nie je pravda. Naivná Bayesovská klasifikácia podáva uspokojujúce výsledky a nachádza svoje uplatnenie v praxi.

V druhej kapitole je vysvetlený základný pojem a zároveň hlavná téma, ktorá sa nazýva získavanie znalostí z databáz a dolovanie dát. Nasleduje jej rozdelenie do procesov a fáz. Vysvetlenie samotnej funkcionality procesu dolovania dát a jeho rozdelenia na rôzne dolovacie úlohy.

V tretej kapitole je presnejšie popísaný konkrétny typ dolovacej úlohy, ktorou je v mojom prípade klasifikácia, ktorá patrí medzi jednu z hlavných typov dolovacích úloh. Obsahuje stručný popis najčastejšie používaných a známych metód, ktoré sa pri klasifikácii vyžívajú.

Vo štvrtej kapitole je popísaná, mnou vybraná, konkrétna klasifikačná metóda, ktorou je naivná Bayesovská klasifikácia. Sú v nej vysvetlené základné princípu, na ktorých si metóda zakladá, ako napríklad podmienená pravdepodobnosť a následne Bayesov vzorec, ktorý je fundamentálnym kameňom celého algoritmu klasifikácie pomocou naivnej Bayesovskej metódy.

Piata kapitola obsahuje popis návrhu aplikácie, ktorá bude demonštrovať proces klasifikácie naivnou Bayesovskou metódou. Použitie rôznych techník a nástrojov, ktoré uľahčovali a sprehľadňovali prácu pri návrhu aplikácie, ale aj jej logickej štruktúry.

V šiestej kapitole je popísaný postup pri implementácii aplikácie. Tu je rozoberaný podrobne algoritmus naivnej Bayesovskej metódy použitej pri klasifikácii, nasledovaný popisom databázovej vrstvy a postup pri implementáciu grafického užívateľského rozhrania, ktoré je dôležitou časťou aplikácie pre interakciu s užívateľom. Tak isto sú v nej spomenuté obmedzenia aplikácie a problémy s ktorými som sa pri implementácii stretol.

Siedma kapitola pojednáva o testovaní aplikácie na trénovanej množine dát a následnom testovaní algoritmu na testovacích dátach. Obsahuje zhrnutie dosiahnutých výsledkov.

Poslednou ôsmou kapitolu je záver, v ktorom je zhrnutý celkový dojem z aplikácie, a celej tvorby tejto práce.

2 Získavanie znalostí z databáz a dolovanie dát

Na začiatku práce bolo nutné pochopiť a naštudovať celú oblasť získavania znalostí z databáz. Pochopiť kľúčové pojmy a štruktúru celého procesu. Nutné bolo pochopiť význam samotných hlavných pojmov získavanie znalostí z databáz a dolovanie dát. Termínom získavanie znalostí rozumieme celý proces, postup, ktorým vrcholom je nejaká znalosť, avšak dolovanie dát tvorí práve jednu časť v tomto procese. V niektorých zdrojoch alebo literatúre je možné sa stretnúť s faktom, že pojmy získavanie znalostí z databáz a dolovanie dát sú brané ako synonymá, je potrebné si to uvedomiť. Tak isto som narazil na asi nie príliš šťastný preklad anglického originálu data mining, teda dolovanie dát. Presnejšie by bolo označenie dolovanie z dát, ak sa zmyslíme nad výsledkom tohto procesu alebo jeho priebehom, nedolujeme dáta, ale informácie resp. vzory z týchto dát, čiže dolovanie z dát. Ide však len o drobnosť nie informatického charakteru, na tento fakt nebude braný žiaden zreteľ. V celej kapitole sa vyskytujú informácie, ktoré som čerpal po preštudovaní literatúry [1], [2].

Môžeme povedať, že získavanie znalostí z databáz je extrakcia (alebo dolovanie) zaujímavých (netriviálnych, skrytých, predtým nepoznaných a potencionálne užitočných) modelov dát a vzorov z veľkých objemov dát. Tieto modely a vzory reprezentujú znalosti získané z dát. Avšak pre dokonalé pochopenie definície, je nutné vedieť správny význam slova „zaujímavých“. Z nášho pohľadu skrýva v sebe niekoľko vlastností. A to informácie, ktoré chceme získať, tie sú, *netriviálne* – znamená že, všeobecne informáciu nie je napríklad možné získať jednoduchým SQL dotazom, *skryté* – znamená že, informácia skrývajúca sa v dátach nám nie je na prvý pohľad zrejmá, nie ju možné vidieť na prvý pohľad, treba sa k nej dopracovať nejakým netriviálnym spôsobom, *potencionálne užitočné* – znamená že, informácia má nejaký význam pre ďalšie rozhodovanie, jej využitie v praxi, predtým *nepoznané* – zaujímajú nás nové informácie. Dolovať takéto informácie môžeme v zásade z akéhokoľvek druhu dát, perzistentných alebo transientých. Asi najčastejším zdrojom dát sú relačné databázy, dátové sklady, transakčné databázy. To však nie sú všetky zdroje menej časté, ale stále možnými zdrojmi sú objektovo orientované databáze, temporálne databázy, textové databázy, rôzne multimediálne databáze, prúdy dát a samotný web ako heterogénna databáza s veľkým potenciálom pre dolovanie.

2.1 Praktické aplikácie získavania znalostí z databáz a dolovania dát

Jednou z oblastí v ktorej dolovanie dát našlo praktické uplatnenie je takzvaná *analýza trhu a marketingu*. Dáta z ktorých získavame potencionálne informácie pochádzajú z databáz, v ktorých sú uložené transakcie kreditnými alebo vernostnými kartami a osobné údaje ich používateľov, ďalej údaje z pokladní o prevedených nákupoch. Príkladom aplikácie na túto oblasť môže byť nájdenie zhlukov, modelových, rovnakých zákazníkov, ktorý majú rovnakú vlastnosť, napríklad na základe výšky mesačného príjmu a rodinného stavu, majú istú tendenciu v rovnakom spôsobe utrácaní peňazí.

Ďalšie uplatnenie si dolovanie dát našlo v takzvanej *analýze nákupného košíku*, ktorej cieľom je nájdenie častých, frekventovaných vzorov, to znamená produkty, ktoré si zákazníci najčastejšie kupujú spolu. Jedná sa o dolovanie takzvaných frekventovaných vzorov a asociačných pravidiel. Asociačné pravidlá vyjadrujú záver pozorovania a analýzy nákupného košíku, ako napríklad zákazník, ktorý si kúpi notebook si zároveň kúpi tašku na notebook a myš k notebooku. V tejto oblasti získavania znalostí môžeme k takejto analýze pridať aj časovú dimenziu, ktorá umožní získavať nejaké časté vzory aj v čase, teda čo a kedy zákazníci najčastejšie nakupujú.

Veľkou oblasťou praktického nasadenia dolovania dát je *finančná analýza a riadenie rizík*. Tu patrí analýza a predikcia vývoja cien v čase, predikcia poskytnutia pôžičiek v bankách a podobne.

V predchádzajúcich odstavcoch sme hľadali informácie na základe nejakých podobných vlastností, ktoré sa vyskytovali v dátach často a hľadali sme príslušné asociácie. No v niektorých prípadoch chceme získať z dát prvý opak, detekovať podozrivé, ojedinelé alebo neobvyklé chovanie. Jedná sa napríklad o oblasť pre *detekciu podvodov a neobvyklého chovania*. Hľadáme vzory, ktoré sa nejakým spôsobom výrazne líšia od častých vzorov. Patria sem také úlohy, ako vyhľadávanie podozrivých bankových transakcií, ktoré môžu indikovať podvod, pranie špinavých peňazí. Tak isto sa môže jednať o vyhľadávanie uzavretých reťazcov z hlásení dopravných nehôd, ktoré môžu signalizovať poistný podvod a podobne. Praktických aplikácií v tomto smere je naozaj veľké množstvo.

Ďalšou oblasťou v ktorej získavanie znalostí našlo uplatnenie je *dolovanie z textových dát*, teda získavanie znalostí z textu. Môže sa uplatňovať pri všetkých prípadoch, v ktorých sú potencionálne informácie uchovávané v textoch, najčastejšie však v emailoch, napríklad pri odhaľovaní spamov. No a v neposlednom rade obrovské množstvá textových dát a informácii sa nachádza na internetových stránkach teda prostredí webu všeobecne. Dajú sa získavať informácie z webových miest o chovaní jednotlivých užívateľov, napríklad pre účel personalizácie a klasifikácie dokumentov.

Do ďalšej oblasti môžeme zaradiť získavanie znalostí v *prúdoch dát*. Zvláštnosťou je, že sa jedná o nasadenie tzv. online analýzy, ktoré spracúva „nekonečný“ prúd analyzovaných dát.

Špecifickou a vyhradenou oblasťou v ktorej sa dá a aj sa aplikuje získavanie znalostí z dát, je rozširujúca sa oblasť bioinformatiky a analýzy biologických dát.

Získavanie znalostí z databáz má istý súvis v dnešnej dobe už s populárnym a známym pojmom *bussiness intelligence*. Týmto pojmom sa označujú metódy, techniky, nástroje a technológie, ktoré slúžia pre podporu rozhodovania. Pri väčšej abstrakcii sem zahrňujeme ale aj osoby, ktoré tieto rozhodnutia vykonávajú. Teda získavanie znalostí obsahuje práve rôzne metódy a techniky, ktoré napomáhajú napríklad pri rozhodovaní.

2.2 Proces získavania znalostí

Celý proces, ktorý sa nazýva získavanie znalostí z databáz, (anglicky knowledge discovery in databases), sa skladá z určitých štádií, ktorých precíznym plnením sa dopracúvame k nejakej znalosti. Prvým štádiom, resp. fázou v procese je *čistenie dát*. To sa prevádza za účelom zbaviť dáta rôznych šumov a nekonzistencií. Napríklad klasické preklepy alebo omyli pri zadávaní údajov do databáz, chýbajúce údaje a podobne. Tieto procesy sú automatizované a bez tejto fázy by výsledná znalosť nemohla byť vierohodná.

Ďalšou fázou procesu je *integrácia dát*, keďže dáta bývajú často uložené vo viacerých databázach, je nutná ich integrácia do jednotného celku.

Nasledujúcou fázou je *výber a transformácia dát*. Zo všetkých dát nás väčšinou zaujímajú len niektoré, napríklad nie všetky atribúty v nejakej relačnej databáze považujeme za potrebné vzhľadom na danú úlohu, tak vyberieme len pre nás relevantné, týmto ušetríme prácu samotným algoritmom, ktoré budú nad týmito dátami realizované. Cieľom transformácie je dáta upraviť, transformovať do vhodnej podoby pre dolovanie, tým, že sa prevedú sumarizačné alebo agregačné operácie. Tieto fázy patria do takzvaného predspracovania dát.

Po týchto fázach vznikli relevantné dáta, tu nastupuje tretia fáza ktorou je *dolovanie dát*, ktorému sa budem venovať podrobnejšie v nasledujúcich kapitolách. V skratke sa dá povedať, že to je základná fáza pri získavaní znalostí, možno práve preto sa dá stretnúť s týmto pojmom ako synonymom pre celý proces teda, získavanie znalostí z databáz. V tejto fáze sú aplikované rôzne metódy, algoritmy pomocou ktorých sa pokúšame získať, vydolovať dátové vzory. Pod slovom vzor sa rozumie nejaká reprezentácia dát, v nejakej forme, to závisí od typu dolovacej úlohy resp. algoritmu. Teda dátovým vzorom môže byť naučená neurónová sieť, množina položiek ktoré majú spolu nejakú súvislosť nie priamo viditeľnú, v nasledujúcich kapitolách sa týmto typom úloh budem venovať podrobnejšie, avšak spomeniem, že v tomto prípade sa jedná o frekventované množiny. Ďalej napríklad rozhodovacie stromy a tak podobne, všeobecné pomenovanie je dátové vzory.

Nezanedbateľnou fázou procesu je *vyhodnocovanie nájdených vzor*. Vzory treba spätne vyhodnotiť na základe mieri ich užitočnosti.

Zostáva samotná *prezentácia* získanej znalosti vo vhodnej forme, zrozumiteľnej pre vedúcich a manažérov. Takto popísaný proces je však v realite len jednou iteráciou, stáva sa, že je nutné vrátiť sa naspať a niektorými fázami procesu prejsť viackrát.

2.3 Dolovanie z dát, funkcionalita dolovania z dát

Keďže poznáme už rôzne zdroje s ktorých môžeme získavať potrebné vzory, zameriam sa na samotnú funkcionalitu dolovania z dát, teda aké vzory môžeme vydolovať. V predchádzajúcej podkapitole sme už nejaké druhy dátových vzorov spomenuli. Čo si vlastne predstaviť pod funkcionalitou dolovania, funkcionalita dolovania z dát nám špecifikuje, aký druh dátových vzorov je možné nájsť pri dolovaní. Vo všeobecnosti je možné tieto úlohy rozdeliť na dve hlavné skupiny: *deskriptívne* a *prediktívne* úlohy. Deskriptívne úlohy charakterizujú všeobecné vlastnosti analyzovaných dát. U prediktívnych úloh na základe analýzy súčasných dát sa dedukuje a predpovedá budúce chovanie.

2.3.1 Rozdelenie dolovacích úloh

Jedným zo základných typov dolovacích úloh je *popis konceptu/triedy*. Dáta sú viazané do určitej triedy alebo tvoria nejaký koncept. Sú dva spôsoby ako tento popis získať. Jedným je charakterizácia dát, čo znamená, že v analyzovanej triede sú dáta sumarizované pomocou výrazov. Takéto dáta sa dajú z databáze získať väčšinou jednoduchým dotazom. Druhý spôsob je diskriminácia dát. Tu nepopisujeme dáta analyzovanej triedy nejakými výrazmi, ale dáta vymedzujeme na základe čo najväčšej odlišnosti atribútov.

Ďalším typom dolovacej úlohy sú takzvané *frekventované vzory*. Ako je zrejmé už zo samotného názvu jedná sa o nejaké vzory, ktoré sa vyskytujú v dátach často. Pre lepšie pochopenie si môžeme predstaviť aplikáciu pre nákupný košík. Fakt, že si zákazník pri kúpe nejakého produktu následne kúpi produkt iný. Uvediem príklad, zákazník má tendenciu si kúpiť digitálny fotoaparát a potom napríklad 4GB pamäťovú kartu, tento fakt nazývame frekventovaným (sekvenčným)

vzorom. Dolovaním takýchto vzorov môžeme dospieť k zaujímavým tzv. asociačným pravidlám a koreláciám. Asociačné pravidlá môžeme rozlišovať na jednorozmerné a viacdimenzionálne asociačné pravidlá na základe počtu predikátov v pravidle.

Iným typom dolovacie úlohy je takzvaná *klasifikácia a predikcia*. Cieľom klasifikácie je nájsť model, ktorý popisuje a zároveň rozdeľuje dáta do tried. Celý proces klasifikácie zostáva z troch hlavných častí, trénovanie klasifikačného modelu, a testovanie modelu nakoľko sme dobre potrénovali a potom použitie takto naučeného modelu na novom vzorku dát, ktorého zaradenie do triedy chceme zistiť. Trénovanie a testovanie prebieha nad dátami ktorých zaradenie do tried poznáme vopred. Pri trénovaní využívame tohto faktu pre určenie klasifikačných pravidiel respektíve vytvorenie modelu podľa ktorého sa bude klasifikovať, v druhej fáze, fáze testovania, takto vytvorený model testujeme, to znamená určenie v ktorých prípadoch model klasifikuje nesprávne. Na základe výsledkov testovania buď model pozmeníme a prevádzame znova trénovanie alebo ho prehlásime za platný a pripravený na použitie. Klasifikácie sa používa pre predikciu kategorických, diskrétnych hodnôt tried, napríklad dobrý klient alebo zlý klient a pod..

Pre hodnoty spojitých atribútov sa používa *predikcia*, znamená to, že v tomto prípade predikujeme chýbajúcu alebo nedostupnú hodnotu.

Ďalšou veľkou oblasťou praktického užitia získavania znalostí je *zhluková analýza*. Hlavným rozdielom od predošlej popisovanej časti klasifikácie a predikcie, ktoré pri vytváraní modelu analyzujú dáta, ktorých rozdelenie do tried poznáme, zhluková analýza analyzuje dáta bez znalosti zaradenia do tried. Teda jej cieľom je nájsť triedy objektov, ktoré majú spoločného čo najviacej. Objekty sa zhlukujú do tried na princípe maximalizácie podobnosti objektov rovnakej triedy a minimalizácii podobnosti objektov odlišných tried. Takto vytvorené triedy majú podobu zhlukov.

3 Klasifikácia a predikcia

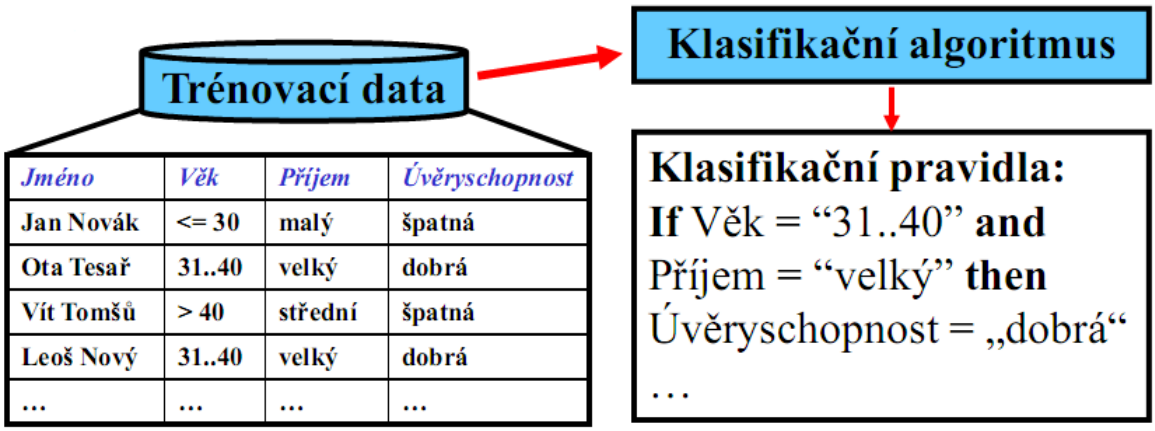
Klasifikácie je všeobecne povedané proces, v ktorom priradzujeme dáta na základe ich vlastnosti do tried konečného počtu. Pre lepšiu predstavu uvediem príklad aplikácie klasifikácie, kde chceme rozdeliť skupinu zákazníkov banky do dvoch skupín. Jedna skupina bude obsahovať zákazníkov s minimálnym rizikom poskytnutia úveru a druhá naopak bude obsahovať tých zákazníkov, u ktorých je poskytnutie úveru rizikovejšie. Existuje však mnoho ďalších príkladov podobného charakteru. Táto kapitola sa opiera o informácie získané z materiálov [2], ktoré som používal pri štúdiu problematiky. Vzorce použité v príslušných podkapitolách sú vybraté z referenčnej knihy [3] pre štúdium a materiálov [2].

Oproti klasifikácii je predikcia proces, ktorý umožní dátam priradzovať hodnoty, ktoré majú všeobecne spojitý charakter. Príklad takejto aplikácie môže byť určenie výšky platu zamestnanca na základe jeho vedomostí, znalostí a vlastností.
Presnejšia charakteristika týchto základných pojmov je nasledovná:

- Klasifikácia – zaradenie daného objektu do triedy na základe jeho vlastností.
- Predikcia – predpoveď hodnoty pre daný objekt na základe jeho vlastností.

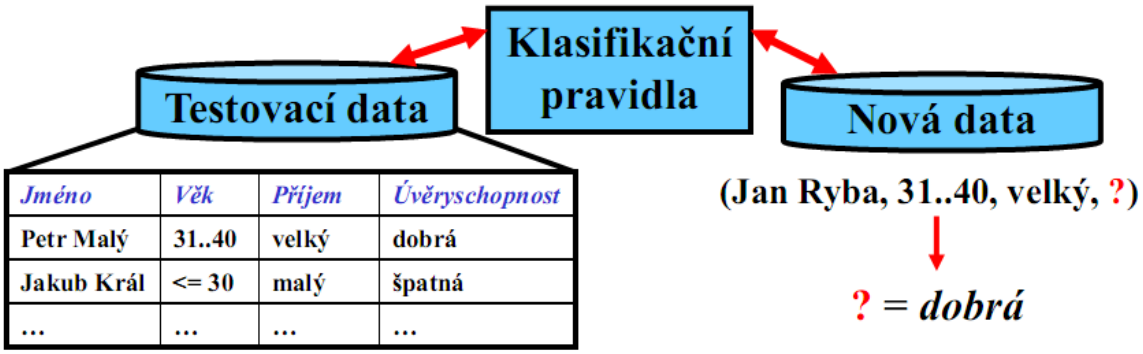
3.1 Klasifikácia ako proces

Klasifikácia prebieha v dvoch hlavných fázach. Prvou fázou je trénovanie klasifikačného modelu alebo taktiež z logického hľadiska učenie modelu, vyberaním vzoriek dát z trénovacej množiny. U týchto dát musíme vopred poznať, do akej triedy sú vzorky zaradené. Tieto vzorky tvoria vstup pre klasifikačný model. Je to algoritmus ktorý nejakým spôsobom (nejakou metódou) zistí klasifikačné pravidlá, pomocou ktorých môžeme s presnosťou klasifikovať vzorky do danej triedy. Pre lepšiu prehľadnosť a predstavivosť je tento proces ilustrovaný na obrázku 3.1.



Obrázok 3.1 Fáza učenia klasifikácie [2] - prevzaté

Druhá fáza je testovanie daného klasifikačného modelu. Tak isto ako v prvej fázy sú vyberané vzorky dát, u ktorých poznáme klasifikačnú triedu dopredu. Tieto vzorky by však mali byť nezávislé na predchádzajúcich dátach, to znamená, že by nemali byť vyberané z trénovacej množiny ale z nejakej novej množiny ktorú nazvime testovacia množina. Ako už bolo spomínané v predchádzajúcich odstavcoch, výstupom prvej fázy sú nejaké klasifikačné pravidlá. V druhej fázy na základe týchto klasifikačných pravidiel priradzujeme dáta z testovacej množiny do tried, pričom vďaka znalosti skutočnej triedy, do ktorej daná vzorka patrí, sa percentami určí, v koľkých prípadoch klasifikačný model správne zaradil danú vzorku. Na základe tohto výsledku sa rozhoduje či je model natoľko „chytrí“ aby mohol byť použitý pre klasifikáciu dát, ktorých zaradenie do triedy nepoznáme a chystáme sa ho práve určiť alebo je potrebné model prepísať alebo nejakým spôsobom upraviť, aby klasifikoval správne. Táto fáza je ilustrovaná na obrázku 3.2.



Obrázok 3.2 Fáza testovania klasifikácie [2] - prevzaté

3.2 Rozdelenie klasifikačných metód

V tejto podkapitole si krátko popíšeme základné klasifikačné metódy. Niektoré z nich sú staršie a už sa v dnešnej dobe neaplikujú alebo len zriedkavo, ako napríklad klasifikácia pomocou rozhodovacích stromov. Ďalej bude spomenutá Bayesovská klasifikácia resp. naivná Bayosovská klasifikácia, ktorej ale bude neskôr venovaná celá kapitola, kde si jej princíp rozoberieme podrobnejšie. Spomenutá bude aj klasifikácia na základe neurónových sietí a metóda SVM (Support Vector Machine), ktorá patrí medze jedny z novších metód. Na konci tejto kapitoly spomenieme ostatné klasifikačné metódy, ktoré nepatria k týmto základným metódam.

Klasifikačné metódy majú svoje charakteristiky, aby bolo možné ich porovnanie. Každá má svoje výhody a nevýhody. Môžeme ich teda charakterizovať na základe niekoľkých vlastností, ktorými sú:

- Presnosť – vyjadruje schopnosť ako kvalitne, správne sú neznáme dáta klasifikované do tried
- Rýchlosť – vyjadruje výpočetnú zložitosť pre dosiahnutie klasifikačných pravidiel
- Robustnosť – vyjadruje schopnosť vytvoriť správny model, ak niektoré dáta obsahujú šum apod.
- Stabilita – vyjadruje schopnosť vytvoriť model pri veľkom objeme dát
- Interpretovateľnosť – vyjadruje zložitosť modelu pre pochopenie

Vo väčšine prípadov však tieto kritériá idú proti sebe. Ťažko nájdeme takú metódu ktorá bude mať najlepšie spĺňať všetky tieto kritériá.

3.2.1 Klasifikácia pomocou rozhodovacích stromov

Rozhodovací strom je graf stromovej štruktúry, kde každý vnútorný uzol reprezentuje test hodnoty atribútu a koncové uzly (listy) reprezentujú triedu, do ktorej je daný objekt klasifikovaný. Takýto rozhodovací strom môže byť ľahko prevedený na odpovedajúce klasifikačné pravidlá.

Najdôležitejšou časťou celej metódy je vybrať atribút, ktorý má najväčšiu rozhodovaciu schopnosť. Existujú tri spôsoby, resp. algoritmy, ktorými sa dá takýto atribút zistiť, ktorá si veľmi skrátka popíšeme.

Prvým spôsobom je použitie *algoritmu ID3*, ktorý v dnešnej dobe patrí k zastaraným a málo používaným. Jeho princípom je počítanie očakávanej informácie $Info(S_1, S_2, \dots, S_m)$, ktorá klasifikuje danú vzorku dát a entropie $E(A)$ pre daný atribút. Odčítaním týchto medzivýsledkov dostaneme výslednú hodnotu $Gain(A)$ ktorá určí, ktorý z atribútov majú najväčšiu rozhodovaciu schopnosť, hľadáme teda najväčšiu hodnotu $Gain(A)$. Pre lepšie pochopenie uvediem vzorce, podľa ktorých algoritmus ID3 funguje. Práca sa však nezaoberá detailmi tejto metódy.

$$Info(S_1, S_2, \dots, S_m) = -\sum_{i=1}^{m} p_i \, log_2 \, (p_i), \qquad E(A) = \sum_{j=1}^{v} \frac{s_{1j} + \ \dots + s_{m\,j}}{|S|} Info\left(s_{1j}, s_{2j}, \dots, s_{mj}\right),$$

kde

$$Info\left(S_{1j}, S_{2j}, \dots, S_{mj}\right) = -\sum_{i=1}^{m} p_{ij} \, log_2 \, (p_{ij}),$$

$$Gain(A) = I(s_1, s_2, \dots, s_m) - E(A)$$

Algoritmus má však nevýhodu v tom, že zvýhodňuje atribúty, ktoré majú veľké množstvá hodnôt.

Tento problém vyriešilo nahradenie *algoritmom C4.5*, ktorý využíva normalizáciu. Výsledná hodnota, ktorá určí vhodný atribút, bude počítaná iným spôsobom, ktorý využíva podiel entropiou pre daný atribút. Takýmto spôsobom prevedieme normalizáciu, ktorá odstráni problém metódy ID3. Entropiu $SplitInfo(A)$ vypočítame podľa vzorca:

$$SplitInfo(A) = -\sum_{j=1}^{v} \frac{|S_j|}{|S|} log_2 \left(\frac{|S_j|}{|S|}\right),$$

a výsledné nové $GainRatio(A)$ spočítame ako podiel, pôvodného $Gain(A)$ a vyššie spočitanej entropie, teda:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

Existuje však ešte jeden spôsob, ktorý je lepší ako predošlé algoritmy. Jedná sa o algoritmus *Gini indexu*. Hlavnou nevýhodou predošlých metód je časté počítanie logaritmov, čo s časového hľadiska nie je dobré. A tento problém vyriešilo použitie práve metódy Gini indexu, ktorá nahradí vzorce používané pri algoritme ID3. Už len v skratke si na ukážku tieto vzorce predstavíme.

$$Gini(S_1, S_2, \dots, S_m) = 1 - \sum_{i=1}^{m} {p_i}^2, \qquad Gini(S_{1j}, S_{2j}, \dots, S_{mj}) = 1 - \sum_{i=1}^{m} {p_{ij}}^2,$$

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + \dots + s_{mj}}{|S|} Gini(s_{1j}, s_{2j}, \dots, s_{mj})$$

Ako je možné vidieť, ide o modifikáciu vzorcov používaných algoritmom ID3. Teda výsledná hodnota, ktorá určí atribút s najväčšou rozhodovacou schopnosť sa spočíta ako:

$$Gini(A) = Gini(S_1, S_2, \dots, S_m) - E(A)$$

Algoritmus, ktorý vytvorí samotný strom, má svoje chyby s ktorými sa ale dá vysporiadať. Pri jeho vytváraní vznikajú vetvy, ktoré znepresňujú výslednú klasifikáciu, je ich za potreby odstrániť. Pre tento prípad sú známe dva postupy.

Prepruning je metóda, ktorá zabezpečí už pri vytváraní rozhodovacieho stromu, aby sa nežiaduce vetvy vôbec nevytvárali.

V prípade metódy postpruning sa najskôr vytvorí celý strom aj s nežiaducimi vetvami a až potom sa z neho odstránia.

Každá z metód má výhody aj nevýhody. Ako už s popisu vyplýva postpruning je výpočetne náročnejšia metóda lebo najprv sa celý strom musí vygenerovať. Avšak táto metóda je spoľahlivejšia ako prepruning. Preto sa v praxi tento fakt často rieši kombináciami týchto metód.

3.2.2 Bayesovská klasifikácia a Bayesovské siete

Jedná sa o iný typ metód, ako v predchádzajúcej podkapitole. Tieto metódy sú založené na pravdepodobnosti. Presnejšie na podmienenej pravdepodobnosti, ktorú poznáme z matematiky. Konkrétne tzv. naivná Bayesovská klasifikácia a Bayesovské siete. O naivnej Bayesovskej klasifikácii bude celá kapitola, v ktorej si túto metódu popíšeme presnejšie. Tu len v krátkosti naznačím, ako už nadpis napovedá bude sa silno využívať princípu Bayesovho vzorca, ktorý vychádza s podmienenej pravdepodobnosti.

Bayesovské siete resp. Bayesovská sieť je orientovaný, acyklický graf, v ktorom každému uzlu grafu pridelím tabuľku podmienených pravdepodobností, ktoré udávajú závislosti jednotlivých atribútov. Pomocou orientovaných hrán určím jednotlivé závislosti. Takýmto spôsobom vznikne sieť.

Klasifikácia prebieha na základe vzorca, podľa ktorého spočítame pravdepodobnosť výskytu vzorku dát, ktorého jeden alebo viacero atribútov je neznámych a tie nám reprezentujú triedy, do ktorých chceme klasifikovať. Vo väčšine prípadov však chceme klasifikovať na základe jedného atribútu. V tomto prípade sa pravdepodobnosť P bude počítať pre jednotlivé atribúty, ale ak sa vyskytujú závislosti atribútov, tak výsledné pravdepodobnosti určí pravdepodobnostná tabuľka daného atribútu, v ktorej sú hodnoty už pred počítané. Takýmto spôsobom vypočítam alebo zistím pravdepodobnosť pre každý atribút a výsledná pravdepodobnosť je súčinom týchto hodnôt. Text môžeme zapísať už spomínaným vzorcom:

$$P(x_1, x_2, \dots, x_n) = \prod_{i=1}^{n} P(x_i | Parents(Y_i))$$

V prípade, že sú neznáme dva lebo viac atribútov, tak za všetky takéto atribúty doplním všetky možné hodnoty, ktorých môžu nadobúdať. A pre každý takýto prípad spočítam pravdepodobnosť, podľa vyššie uvedeného vzorca a vyberiem práve tie atribúty pre ktoré je výsledok najväčší a do tohto atribútu budem klasifikovať. Veľkým problémom tejto metódy je jej interpretovateľnosť. Vytvoriť takýto model je veľmi zložité. Je potrebné vytvoriť topológiu siete tzn. jednotlivé závislosti atribútov a ďalej treba nejakým spôsobom zistiť jednotlivé pravdepodobnostné tabuľky patriace atribútom.

Táto metóda vznikla ako odpoveď na nedostatok práve naivnej Bayesovskej klasifikácie. Vyriešili problém naivity Bayesovkej klasifikácie, ktorá predpokladá pri klasifikácii, že jednotlivé atribúty sú na sebe nezávislé, v praxi často toto tvrdenie neplatí. Práve preto si Bayesovská klasifikácia zaslúžila prívlastok naivná. Ale existujú prípady, kde sa nasadzuje a podáva uspokojivé výsledky, hlavne v prípadoch, keď sú atribúty skutočne nezávislé.

3.2.3 Klasifikácia pomocou neurónových sietí

Najjednoduchšou sieťou je jednoneurónová sieť tzv. *Perceptron*. Ten sa skladá zo vstupov resp. vstupných dát, ktoré označíme $x_1, x_2, \dots, x_n$ a z hrán, ktoré majú váhy označíme $w_1, w_2, \dots, w_n$. Tieto hodnoty si možme predstaviť ako dva vektry o dĺžke n a klasickým skalárnym súčinom dostaneme jednu výslednú hodnotu. Neuron má tak isto ešte telo tzv. bias, čo je konštanta ktorá sa pripočítam k vstupnej hodnote. V praxi sa však často toto pričítanie zjednoduší, že hneď na prvý vstup privedieme jednotku, to znamená, že bias je už nastavený vo váhe w_1. A tejto celej časti neurónu sa hovorí bázová funkcia. Tá je vstupom do aktivačnej funkcie, s ktorej je potom jeden výstup.

S pohľadu klasifikácie Perceptron vie klasifikovať len do dvoch tried a navyše dáta musia byť lineárne separovateľné tzn. z hľadiska roviny je môže vytvoriť priamku, ktorá rozdelí rovinu, ktorá predstavuje dáta práve do dvoch tried. Využíva sa rovnice priamky z matematiky, ktorá predstavuje klasifikáciu dát.

$$w_1 + w_2 x + w_3 y = 0$$

Ak vyjde kladné číslo klasifikujeme do triedy jedna, ak záporné do triedy mínus jedna. Toto platí pri aktivačnej funkcii Perceptronu, $y = sgn(x)$, ktorá práve vráti jednotku, pre celkový kladný výsledok alebo zápornú jednotku, ak je celkový výsledok záporný.

Učenie klasifikačného modelu pri tejto metóde predstavuje na začiatku nastavenie ľubovoľných váh, potom sa iteratívne prechádzajú dáta pokiaľ sieť nebude naučená. Ak budú vzorky klasifikované správne – to zistíme dosadením do rovnice, tak sa váhy nemenia v opačnom prípade, ak mala byť vzorka klasifikovaná do triedy jedna, tak hodnotu klasifikovaného prvku k váham pripočítame, ak mala byť vzorka klasifikovaná do triedy mínus jedna tak túto hodnotu odčítame. Tento proces sa iteratívne opakuje až model bude klasifikovať správne.

Zložitejšou sieťou skladajúcu sa z viacero neurónov je tzv. neurónová sieť *Backpropagation*. Delí sa na tri časti, vstupná vrstva, ktorá predstavuje dáta ktoré chceme klasifikovať – každému neurónu odpovedá jeden atribút vzorku dát. Skryté vrstvy v ktorých sa počíta a učí sieť a posledná výstupná vrstva s ktorej vychádza výstup. Ak však chcem klasifikovať všeobecne do k tried výstup tvorý k neurónov. Sieť dostane na vstupnú vrstvu vstupné dáta pre každý neurón, ktorý len rozdistribuuje tieto hodnoty ďalšej vrstve, ktorá spočíta hodnoty tak ako bolo vysvetlené

u Perceptronu a na výstup dostaneme výsledky, ak klasifikoval správne proces končí. Učenie takejto siete prebieha v prípade zlej klasifikácie na princípe spätného šírenia chýb naspať od výstupnej vrstvy po neurónoch. Podľa toho kde a ako veľmi sieť klasifikovala zle, sa znova nastavia váhy a proces učenia sa znova opakuje pokiaľ neklasifikujeme správne – vtedy je sieť naučená a model funguje správne. Podľa tohto princípu dostala sieť meno Backpropagation. Celý proces môžeme tak isto rozdeliť do troch častí. Inicializácia, podobne ako u Perceptronu, ľubovoľne nastavím váhy a biasy všetkým neurónom. Šírenie vstupu na výstup, po spočítaní skalárneho súčinu pre daný neurón a pripočítaním biasu dostaneme medzivýsledok, ktorý dosadíme do aktivačnej funkcie, ktorej výsledok je pre nás výstupom. Tieto hodnoty zistíme aplikovaním nasledovných vzorcov:

$$I_j = \sum_i w_{ij}\, O_i + \theta_j$$

Všeobecne zapísaný spôsob skalárneho súčinu váh a pričítanie biasu, tým získame medzivýsledok.

$$O_j = \frac{1}{1+e^{-I_j}},$$

predstavuje aktivačnú funkciu každého neurónu. Spätné šírenie chyby potom funguje na základe vzorcov. Pre každý neurón výstupnej vrstvy spočítame chybu:

$$Err_j = O_j\big(1 - O_j\big)(T_j - O_j),$$

kde O_j je náš výstup a T_j predstavuje výstup, ktorý by mal vyjsť. Pre každý neurón skrytej vrstvy spočítame chybu:

$$Err_j = O_j\big(1 - O_j\big)\sum_k Err_k w_{jk},$$

potom pre každú váhu sa vypočíta nová hodnota Δw_{ij} v závyslosti na spomínanej chybe ku ktorej sa ešte pridá tzv. koeficient učenia l, ktorý je vhodné zvoliť ako $\frac{1}{t}$, kde t je počet iterácií:

$$\Delta w_{ij} = (l) Err_j O_i,$$

tú pripočítame k pôvodnej hodnote váhy:

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

Rovnakým spôsobom spočítame aj hodnoty nových biasov $\Delta\theta_j$ a pripočítame k pôvodným:

$$\Delta\theta_j = (l) Err_j, \qquad \theta_j = \theta_j + \Delta\theta_j$$

3.2.4 Support Vector Machine (SVM)

Je nová klasifikačná metóda, v jej základnej verzii podobná neurónovej sieti Perceptron. Rozdiel je v hľadaní hranice, ktorá by dáta rozdeľovala čo najviacej od seba. Pre klasifikáciu sa používajú dve rovnice priamky, čiastočne modifikované, ktoré práve tento pás resp. hranicu vymedzia.
Do prvej triedy sa budú klasifikovať dáta, splňujúce túto podmienku:

$$w_0 + w_1 x_1 + w_2 x_2 \geq 1,$$

do druhej triedy:

$$w_0 + w_1 x_1 + w_2 x_2 \geq -1$$

Metóda sa dokáže vysporiadať aj s nelineárnymi dátami na rozdiel od Perceptronu, to znamená dáta, ktoré sa nedajú separovať priamkou. Tieto dáta sa linearizujú a to pridaním nových atribútov a tým pádom sa zmení ja funkcia, pomocou ktorej klasifikujeme.

3.2.5 Ostaté druhy klasifikácie

Existujú aj iné metódy využívane pri klasifikácii. Známy je napríklad klasifikačný model založený na *k-najbližšom susedstve* v ktorom je každá vzorka dáta reprezentovaná istou n-ticou atribútov číselnej hodnoty. Jednotlivé tvoria n-dimenzionálny priestor. Medzi dvomi vzorkami $X = (x_1, x_2, \dots, x_n)$ a $Y = (y_1, y_2, \dots, y_n)$ je definovaná Euklidovská vzdialenosť, ktorá sa počíta pomocou vzorca:

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

V rámci klasifikácie sa vyberú z trénovacej množiny práve tie vzorky, ktorých vzdialenosť je najmenšia vzhľadom ku klasifikovanej vzorke. Tá je potom klasifikovaná do triedy, ktorá je najčastejšia u prvkov vybraných z trénovacej množiny.

Ďalšími dvomi možnými metódami sú *klasifikátory založené na generických algoritmoch* a *klasifikátory založené na fuzzy množinách*.

4 Naivná Bayesovská klasifikácia

Táto metóda ako už bolo spomínané, sa opiera o podmienenú pravdepodobnosť, z ktorej vychádza Bayesov vzorec. Vzorce použité v tejto kapitole som naštudoval z literatúry [3]. Pravdepodobnosť nejakého javu X, je hodnota vyčísľujúca istotu resp. neistotu výskytu určitej udalosti, zapisujeme to ako $P(X)$. Podmienená pravdepodobnosť je hodnota vyčísľujúca istotu resp. neistotu výskytu určitej udalosti v závyslosti k nejakej podmienke. Jav X môže nastať len vtedy, ak nastal jav Y, vtedy hovoríme o podmienenej pravdepodobnosti javu X. Tento vzťah sa zapisuje ako $P(X|Y)$, platí:

$$P(X \cap Y) = P(X|Y)P(Y)$$

Zápis vyjadruje pravdepodobnosť, že navzájom na seba závislé javy X a Y nastanú súčasne. Tak isto platí vzťah:

$$P(X \cap Y) = P(X|Y)P(X)$$

v našom prípade chceme vyjadriť a počítať práve podmienenú pravdepodobnosť, teda si z rovníc vyjadríme $P(X|Y)$ a dostaneme spomínaný Bayesov vzorec:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

Tohto princípu využíva práve metóda naivnej Baysovskej klasifikácie. Ak si značenie prevedieme pre nás do zmysluplnejšej formy, teda:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

kde

- X – predstavuje vzorku dát
- C_i – triedu do ktrorej klasifikujeme

tak chceme určiť s akou pravdepodobnosťou vzorka dát X patrí práve do triedy C_i. Teda hľadáme prípad, pre ktorú triedu je hodnota $P(C_i|X)$ najväčšia. Toto je hlavným princípom tejto klasifikačnej metódy. Podľa Bayesovho vzorca, tejto hodnoty môžeme dosiahnuť a s toho vyplýva, že nám stačí spočítať hodnotu v čitateli, teda $P(X|C_i)P(C_i)$, kedže $P(X)$ je nejaká konštanta.

$P(C_i)$ vyjadruje pravdepodobnosť, že ľubovolne zvolený prvok patrí do tredy C_i. Hodnotu získame podľa nasledujúceho vzorca:

$$P(C_i) = \frac{s_i}{S}$$

kde

- s_i – je počet vzoriek v triede C_i
- S – je počet všetkých vzoriek

$P(X|C_i)$ vyjadruje pravdepodobnosť, že ľubovoľne vybratý prvok zaradený do triedy C_i, bude mať rovnaké hodnoty atribútov ako prvok X. Pre každú hodnotu atribútu x_k patriacej do triedy C_i spočítame pravdepodobnosti a spravíme ich súčin.

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

kde hodnotu $P(x_k|C_i)$ určíme na základe typu atribútu x_k takto:

- atribút x_k má diskrétny charakter

  $$P(x_k|C_i) = \frac{s_{ik}}{s_i}$$

  kde
  - s_{ik} - je počet vzoriek z triedy C_i, ktorých hodnota $k - teho$ atribútu je rovná x_k
- atribút x_k má spojitý charakter

  $$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

  kde
  - $g(x_k, \mu_{C_i}, \sigma_{C_i})$ – je Gausovo normálne rozloženie pre atribút A_k s hodnotou x_k, priemernou hodnotou μ_{C_i} a smerodajnou odchýlkou σ_{C_i}, patriacich do triedy C_i.

V našom prípade, a mojej aplikácii, budeme predpokladať len s diskrétnymi hodnotami atribútov, ale táto metóda dokáže všeobecne počítať aj so spojitými hodnotami.

Úlohou tohto klasifikačného modelu je, vypočítať pravdepodobnosti pre všetky vzorky v jednotlivých atribútoch. Výsledky pre jednotlivé vzorky tvoria takzvané klasifikačné pravidlá. Tento proces teda nazývame trénovanie resp. učenie klasifikačného modelu, tvorí prvú fázu. Druhá fáza je samotná klasifikácia, čo pri tejto metóde predstavuje násobenie pravdepodobností.

Veľkou výhodou tejto metódy je jej robustnosť. Tak ako predošlé metódy však má tiež isté problémy, ktoré treba riešiť. Problém môže nastať, ak podmienená pravdepodobnosť $P(x_k|C_i) = 0$, to znamená (vďaka násobeniu), že $P(X|C_i) = 0$ čo znehodnotí celú klasifikáciu. Riešením je *Laplacelova korekcia*, čo znamená pridanie jedného prvku do všetkých množín. No stále hlavnou nevýhodou je už ako bolo spomínané naivné predpokladanie nezávislosti atribútov.

5 Návrh aplikácie

Pred začatím samotnej implementácie sa bolo treba zamyslieť nad tým, ako bude aplikácia používaná, navrhnúť logickú štruktúru s ohľadom na jej použitie. To znamená s pohľadu objektovo orientovaného programovania, rozdeliť aplikáciu do tried, premyslieť komunikáciu medzi jednotlivými neskôr vytvorenými objektmi. Navrhnúť štruktúru jednotlivých tried, to znamená aké dáta resp. hodnoty potrebujem uchovávať a aké operácie budem potrebovať nad objektom realizovať.

Snažil som sa použiť niektoré vhodné návrhové vzory, avšak v skutočnosť som využil a v aplikácii použil len návrhový vzor singleton pre triedu, ktorá má na starosti prácu s databázou. Jeho úlohou je zabráneniu vytvárania nových inštancií triedy. V celej aplikácii tým pádom je vytvorený len jeden objekt tejto triedy. Ďalej som sa inšpiroval návrhovým vzorom MVC (Model-View-Controler) ktorého použitím som chcel dosiahnuť oddelenia aplikačnej logiky od dáta, avšak sa mi vzor nepodarilo úplne dodržať je veľmi zjednodušený. Pri návrhu som využil len rozdelenia aplikácie do troch hlavných tried resp. vrstiev, databázová, logická a GUI (anglicky Graphical User Interface). Pri návrhu som dbal na ľahkú prístupnosť a ovládanie celej aplikácie. Preto som sa snažil zvoliť vhodné užívateľské rozhranie, ktoré je spracované graficky.

Databázová vrstva bola navrhovaná aby dokázala spracovávať atribúty pre prihlasovanie k databáze. Metódy, ktoré sprístupňujú prácu s databázou, sa umožňujú dotazovať na databázu. Je tým pádom úplne odtienená od aplikačnej logiky.

Vrstva aplikačnej logiky obsahuje triedu, ktorá tvorí jadro celej aplikácie obsahuje atribúty, v ktorých uchovávam výsledky a medzivýsledky výpočtov a metódy, ktoré zabezpečujú celý výpočetný proces na základe dát z databázovej vrstvy.

Grafické užívateľské rozhranie plní svoju úlohu pri interakcii užívateľa s celým procesom. Ponúka mu jednoduchý prístup k zadávaniu hodnôt pre prácu s predošlými vrstvami. Tak isto prezentuje prehľadný výstup, ktorý užívateľ očakáva.

6 Implementácia

Kapitola nás prevedie celou implementáciou aplikácie, podrobnejšie si rozoberieme popis jednotlivých tried z implementačného hľadiska. Bude obsahovať popis a vysvetlenie jednotlivých metód. Ďalej bude spomenutý ako prebieha proces spustenia a fungovanie aplikácie, čiže vzájomná komunikácie medzi jednotlivými triedami.

Aplikácie je implementovaná za použitia programovacieho jazyka Java, vývojového prostredia NetBeans a databázového systému MySQL. Pre komunikáciu s databázou som použil oficiálny JDBC ovládač Connector/J. Pre vytváranie nových databáz a tabuliek, pridávania a prípadne importovania záznamov do databázy som využil aplikácie phpMyAdmin, ktorá spomenuté operácie poskytuje. Aplikácia phpMyAdmin je dostupná v rôznych balíčkoch, ktoré väčšinou už obsahujú databázový server, ako napríklad MySQL, ktorý práve moja aplikácia využíva.

6.1 Špecifikácia

Mojou úlohou bolo naimplementovať naivnú Bayesovskú klasifikačnú metódu, overiť jej činnosť a previesť experimenty na vhodne zvolenom vzorku dát. Pri plnení tejto úlohy som proces implementácie rozdelili do niekoľkých krokov. Vychádzal som z návrhu aplikácie v ktorom som spomenul, že celá štruktúra sa dá rozdeliť do troch hlavných vrstiev s toho faktu som vychádzal aj pri implementácii. Jadrom celej aplikácie je klasifikačný algoritmus, ktorého vstupom sú informácie získané a spracovávané databázovou vrstvou. Preto bolo potreba dbať na vhodné spracovávanie týchto dát a následne vhodné zobrazenie výsledkov užívateľovi, čo bolo súčasťou návrhu a implementácie grafického užívateľského rozhrania.

6.2 Databázová vrstva

Úlohou tejto vrstvy ako už bolo spomínané, je zabezpečiť komunikáciu a získavanie potrebných informácii z dát uložených v databáze. Celá vrstva sa skladá z jednej hlavnej triedy s názvom *Database*, ktorá je uložená v súbore *Database.java*. Tento súbor je súčasťou zdrojového balíčka nazvaného *bp.model*, ktorého súčasťou je tiež trieda *MyKey* uložená v súbore s názvom *MyKey.java*.

Táto hlavná trieda *Database* je implementovaná ako Singleton, to znamená, že inštancia tejto triedy je vytvorená zároveň pri inicializácii triedy signletonu a obsahuje metódu, ktorá vráti inštanciu tejto triedy.

Pre zahájenie spojenia aplikácie s databázou je práve v tejto triede metóda, ktorá tento proces umožní. Jedná sa o metódu `makeConnection( )`, ktorá na základe zadaných prihlasovacích údajov prevedie prihlásenie do databázy. Opakom je metóda `closeConnection( )`, ktorá ukončí spojenie s databázou.

Ďalej trieda obsahuje jednotlivé metódy, ktoré nad dátami v databáze vykonávajú operácie potrebné ku klasifikácii. Všetky nasledujúce metódy sú založené na princípe dotazovania sa na databázu pomocou jazyku SQL. Každá z nich predstavuje iný typ dotazu a každá má ako parameter názov tabuľky do ktorej sa dotazujem. Prípade ďalšie parametre potrebné na vytvorenie požadovaného SQL dotazu v závislosti na úlohe danej metódy.

Menovite, `getRowsCount(String tableName)`, ktorá spočíta a ako výsledok vráti počet všetkých záznamov v danej tabulke, ktorej názov je parametrom metódy. Podobnou metódou je `getColumnCount(String tableName)`, ktorá naopak vráti počet stĺpcov tabuľky.

Metóda `getTableAttributes(String tableName)` vráti pole reťazcov, v ktorom sú uložené názvy stĺpcov danej tabuľky.

Pri klasifikácii všeobecne rôznych dát, to znamená aplikácia dokáže operovať nad ľubovoľnými správne predpracovanými a upravenými dátami uloženými v databázy, bolo potrebné zistiť počet do koľkých tried klasifikujem nové dáta. Túto funkcionalitu zabezpečí metóda `getClassCount(String tableName)`.

Následne potrebujem vedieť, koľko záznamov je celkovo klasifikovaných do jednotlivých tried, ak viem názov tabuľky nad ktorou chcem tento dotaz aplikovať, pole v ktorom sú názvy stĺpcov. V tomto prípade však dotaz vyberie len prvý stĺpec v ktorom ukladám triedy do ktorých sa klasifikuje ak viem počet tried. Pre tento účel je vytvorená metóda `getSpecificClassCount(String tableName, String[] attributes, int classCount)`, ktorá obsahuje SQL dotaz, ktorým zistím koľko záznamov je priradených do jednotlivých tried.

Ďalším vstupom pre klasifikačný model je hodnota, ktorá určuje koľko riadkov tabuľky patrí do jednotlivých tried a zároveň ma určitú hodnotu v stĺpci. Dotaz obsiahnutý v tejto metóde vyberie jednotlivé hodnoty z prvého stĺpca (stĺpec v ktorom uchovávam triedy) a bude prechádzať jednotlivo zbytok stĺpcov tabuľky a počítať koľko hodnôt v danom stĺpci patrí konkrétne do jednej z tried. Takýmto spôsobom prejdem celú tabuľku a výsledky si uložím do kolekcie `Map`, ktorú poskytuje programovací jazyk Java. Túto funkcionalitu obsahuje metóda `getSpecificCount(String tableName, String[] attributes)`.

Pre jednoduché vyhľadávanie týchto číselných hodnôt som vytvoril triedu `MyKey`, pomocou ktorej ukladám alebo vyhľadávam tieto hodnoty v mape. Trieda rieši akým spôsob môžem tieto hodnoty uložiť na presnú pozíciu kvôli spätnému vyhľadávaniu. Obsahuje práve dva reťazce určujúce presne, prvý hodnotu triedy, druhý hodnotu v niektorom zo stĺpcov a to v ktorom konkrétne určí práve posledná číselná hodnota udávajúca číslo stĺpca.

Metóda `getCountInClassAttribute(String tableName, String[] attributes)` funguje na rovnakom princípe ako predošlá metóda s tým rozdielom, že vráti počet koľko riadkov patrí do jednotlivých tried a tieto počty uloží do mapy ktorú indexujem práve názvom danej triedy.

Poslednou metódou, ktorá vráti pole, v ktorom sú hodnoty tabuľky zadanej parametrom, na nami zadanom stĺpci, ktorý je druhým parametrom je `getArrayOfValues(String tableName, String attribute)`.

6.3 Vrstva aplikačnej logiky

Vrstva obsahuje triedu `ClassificationClass` uloženú v súbore *ClassificationClass.java* a implementovanú v zdrojovom balíčku `bp.classification`, ktorá zabezpečuje hlavný proces klasifikácie. Vstupom pre klasifikačný model sú predpočítané hodnoty z databáze. Implementácia triedy sa opiera o teoretický postup rozdelený do dvoch fáz. V praxi som však zistil, že tieto fázy môžu byť implementované v rámci jednej metódy. Tento proces obsahuje metóda `training( )`. Tá pre svoju potrebu využíva metódu `addIndex(int index)`, ktorej úlohou je napomáhať pri indexovaní hodnôt tabuľky. Pre výslednú pravdepodobnosť potrebujem ešte vypočítať pravdepodobnosť rozdelenia trénovacích dát do tried. Túto hodnotu vypočítam v metóde

`getEssentialProbability(int[] values, int classCount, int rowsCount)`. Presný postup implementácie bude zahŕňať časté odkazy na metódy v predošlej databázovej vrstve, keďže vstupom pre trénovanie modelu sú práve hodnoty a výpočty z databáze.

Úplne na začiatku však zavediem konvencie pre kľúčové premenné triedy `ClassificationClass`. Jedná sa o hešovaciu mapu `mapOfClass`, ktorá slúži pre uchovanie počtu záznamov resp. riadkov klasifikovaných do jednotlivých tried. Hodnoty v mape uložené sú tým pádom čísla, ale pristupujem k ním na základe indexu, ktorým je reťazec určujúci presne o akú konkrétnu triedu sa jedná. Tieto reťazce sú uložené v poly `classValues`, pomenujem ho ako pole tried. Tento počet zistí metóda `getCountInClassAttribute(String tableName, String[] attributes)` z databázovej vrstvy.

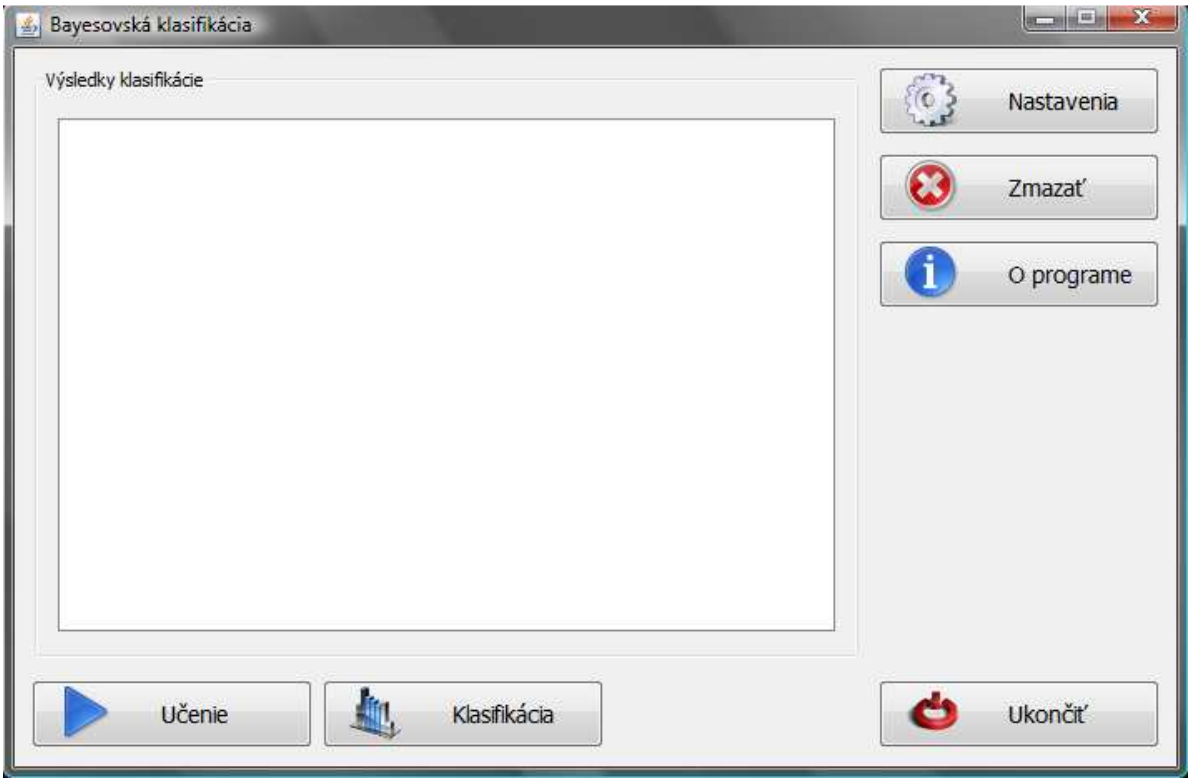
Druhou premennou je hešovacia mapa `map`, ktorá uchováva číselnú hodnotu, ktorú vypočítame hneď na začiatku metódou `getSpecificCount(String tableName, String[] attributes)` z databázovej vrstvy. Keďže metóda vráti celú mapu, nastáva otázka, ako sú výsledné hodnoty v mape uložené v prípade, že sa k ním chcem dostať. Hodnota v mape uložená leží na indexe tvorenom špeciálnym kľúčom. Pre tento účel som práve vytvoril triedu `MyKey`. Výsledky sú do mapy v rámci spomínanej metódy hneď ukladané pomocou tohto kľúča. Ten tvorí hodnota z pola tried, hodnota z pola v ktorom sú ostatné hodnoty atribútu resp. stĺpca určujúceho práve tretím parametrom, ktorým je index stĺpca. Metóda prevedie a spočíta všetky kombinácie.

S takto spočítanými hodnotami môžem začať s počítaním pravdepodobnosti $P(x_k|C_i)$ pre všetky kombinácie jednotlivých hodnôt v spomínaných dvoch poliach. Postupne sa prechádzajú hodnoty uložené v mape `map` na základe indexu stĺpca, reťazca uloženého v poly tried a reťazce uloženého v poly ostatných hodnôt stĺpcov. Tieto hodnoty tvoria čitateľ pomyselného zlomku. Menovateľ práve tvorí hodnota z mapy `classValues`, ktorá je indexovaná rovnako ako druhý parameter z mapy `map`. Týmto spôsobom je zaručené že mám v podiely správne hodnoty, ktoré ďalej na základe rovnakého kľúča ukladám naspäť do mapy. Preto aby som zistil výslednú pravdepodobnosť $P(X|C_i)$ pre všetky riadky, musím ponásobiť hodnoty z jednotlivých stĺpcov. Výsledky násobenia sa ukladajú do novej mapy `preResultMap`, a zároveň v tomto procese vytváram nový kľúč, podľa ktorého prebieha samotná klasifikácia. Celý tento proces tvorí telo metódy `training( )`. Spomínaný kľúč tvorí reťazec, zložený s hodnôt uložených v jednotlivých stĺpcoch oddelenými znakom pomlčka. Za týmto znakom sa pre finálny výpočet pridá hodnota zo stĺpca resp. pola tried. A práve na základe tejto poslednej hodnoty, ktorý tvorí finálny kľúč prinásobím k tejto hodnote z mapy `preResultMap`, pravdepodobnosť rozdelenia všetkých dát z tabuľky do jednotlivých tried. Hodnotu tejto pravdepodobnosti mi vráti metóda `getEssentialProbability(int[] values, int classCount, int rowsCount)`. Týmto definitívne skončil proces klasifikačného algoritmu výsledok sa uloží do mapy `final`.

6.4 Grafické užívateľské rozhranie

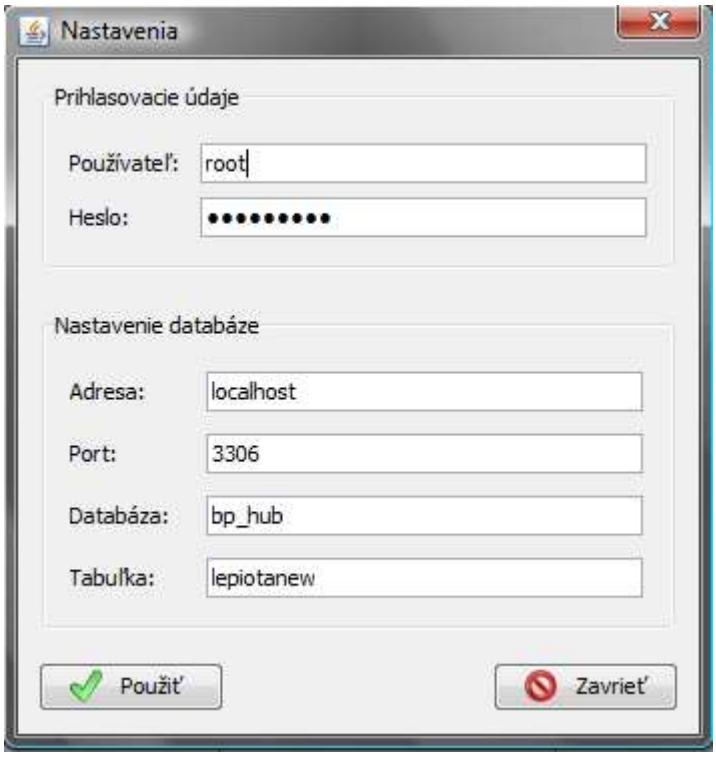
Pre pohodlnú manipuláciu s programom bolo vytvorené grafické užívateľské rozhranie. Je implementované v zdrojovom balíku nazvanom `bp.gui`. Ten obsahuje štyri základné triedy. Hlavným ovládacím prvkom je okno aplikácie, ktoré sa zobrazí po jej spustení.

Jedná sa o triedu `MainWindow`, uloženú v súbore *MainWindow.java*. Trieda dedí od objektu `javax.swing.JFrame`. Obsahuje základné rozhranie, tlačidla pre spustenie procesu klasifikácie, nastavenia databáze, vymazania plochy, v ktorej sa zobrazujú výsledky, tlačidlo pre zobrazenie informácií o aplikácii a tlačidlo, ktoré celú aplikáciu ukončí. Pre lepšiu názornosť môžeme hlavné okno aplikácie vidieť na obrázku 6.4.1.



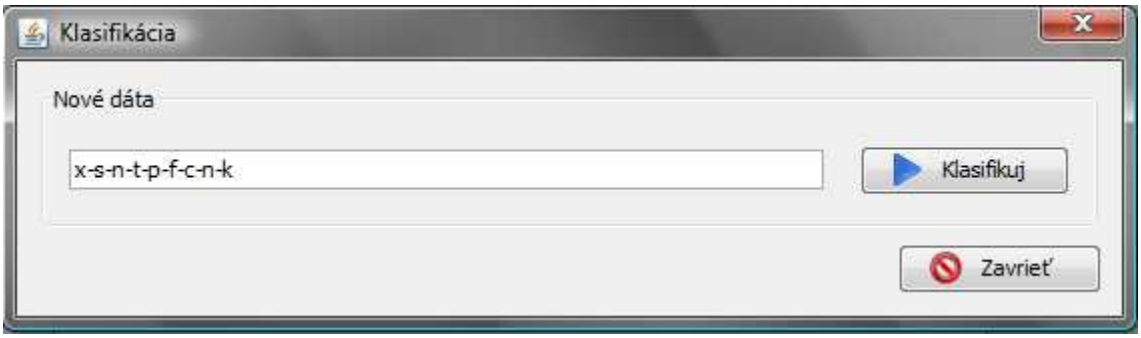
Obrázok. 6.4.1 Hlavné okno aplikácie po jej spustení

Ďalšou triedou je trieda s názvom `SettingsWindow`, uložená v súbore *SettingsWindow.java*. Tvorí rozhranie pre nastavenie pripojenia do databáze. Obsahuje textové polia pre rôzne prihlasovacie údaje a pole určené pre zadanie prihlasovacieho hesla. Aby bolo možné tieto informácie nejako použiť alebo okno zavrieť, sú v triede implementované dve funkčné tlačidlá. Jedným informácie použijem na prihlásenie a zároveň pripojenie do databázy, druhým okno zavriem. Trieda dedí od objektu `javax.swing.JDialog`, teda nie je klasickým oknom, avšak pre potreby, ktoré trieda plní je práve vhodné použitie triedy `JDialog`. Okno môžeme vidieť na obrázku 6.4.2.



Obrázok 6.4.2 Nastavenia

Trieda `ClassifyWindow` predstavujúca rozhranie, cez ktoré môže užívateľ zadávať nové dáta pre klasifikáciu. Trieda dedí z objektu `javax.swing.JDialog` a je uložená v súbore *ClassifyWindow.java*. Obsahuje textové pole do ktorého užívateľ zadá potrebné dáta a aby spustil proces vyhľadania správneho výsledku je v triede pripravené funkčné tlačidlo, ktoré túto službu zaobstará. Následne sa do hlavného okna aplikácie v časti pre zobrazenie výsledkov vypíšu výsledky klasifikácie. Trieda má aj druhé funkčné tlačidlo pre prípadné zatvorenie okna. Názorne ukázané na obrázku 6.4.3.

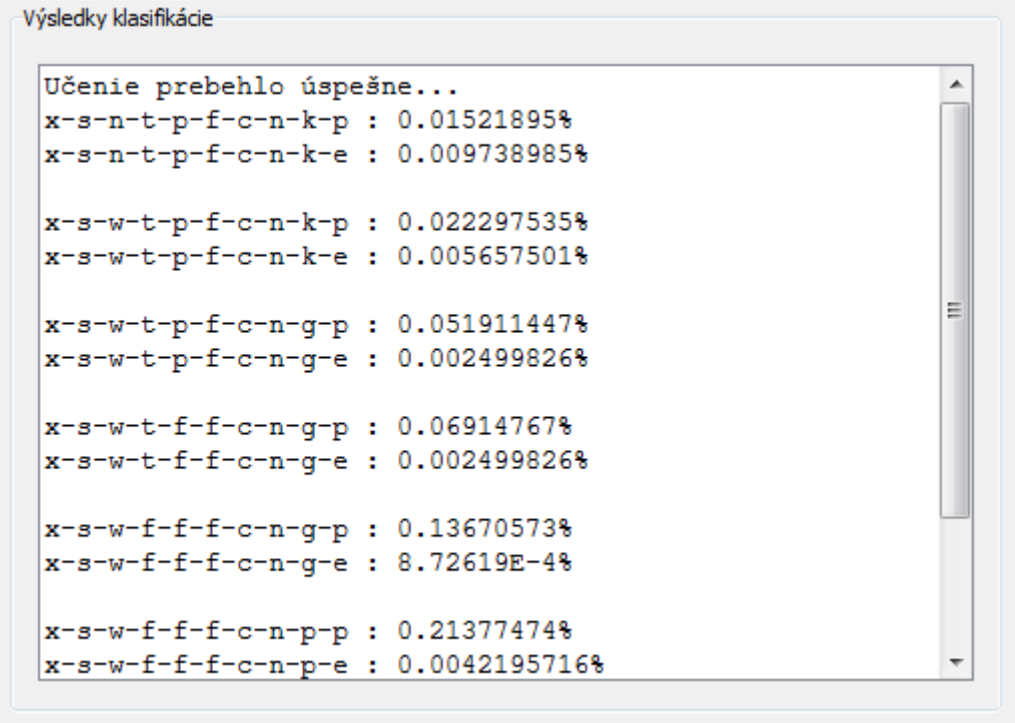


Obrázok 6.4.3 Okno pre zadanie nových dát pre klasifikáciu

Poslednou triedou je trieda `AboutWindow` uložená v súbore *AboutWindow.java*, ktorá predstavuje dialógové okno v ktorom sú informácie o programe.

6.5 Proces klasifikácie

Samotný proces klasifikácie, začína po prihlásení do databáze pomocou tlačidla nastavenia v hlavnom okne aplikácie. Po zadaní prihlasovacích údajov a údajov týkajúcich sa konkrétnej databáze a dát nad ktorými chceme klasifikáciu prevádzať, prevedieme pripojenie k databáze stlačením tlačidla použiť. Po jeho stlačení získam inštanciu triedy `Database` a všetky vyplnené údaje sa uložia v príslušnej triede. Túto funkcionalitu zabezpečí metóda `setUserInfo( )` a následne sa zavolá metóda `makeConnection( )`, ktorá prevedie samotné prihlásenie na databázový server.
Po úspešnom prihlásení sa musí klasifikačný model naučiť na konkrétnych dátach. Jeho výstupom budú klasifikačné pravidlá, podľa ktorých sa budú nové dáta klasifikovať. Učenie modelu sa spustí stlačením tlačidla učenie v hlavnom okne aplikácie. Po jeho stlačení sa vytvorí inštancia triedy `ClassificationClass`, ktorá ako už bolo spomínane a názov sám napovedá rieši túto funkcionalitu. Overí sa , či sme pripojený k databáze, ak áno, zavolá sa metóda `training( )`, ktorá prevedie učenie klasifikačného modelu a zistí klasifikačné pravidlá. Tie sa uložia do kolekcie a čakajú na proces zadania nových dáta. Tie sa zadávajú po stlačení tlačidla klasifikácia v hlavnom okne aplikácie. Po jeho stlačení je možné v dialógovom okne zadať nové dáta v korektnej forme. Správnosť overím po stlačení tlačidla klasifikuj, ak v prípade korektného zadania sa prevedie proces vyhľadania konkrétnych hodnôt v spomínanej kolekcii. Výsledné hodnoty ale aj nové dáta, pre ktoré sme zisťovali klasifikačnú triedu vypíšem v prehľadnej forme do textového poľa v okne hlavnej aplikácie. Na základe výsledných pravdepodobností pre jednotlivé triedy sa dá určiť, do ktorej triedy budú dáta klasifikované. Výsledky klasifikácie vyzerajú v podobe ktorá je na obrázku 6.5.1.



Obrázok 6.5.1 Výpis výsledkov klasifikácie nových dát

6.6 Obmedzenia a možné rozšírenia

Aplikácia nerieši prácu s databázou, ako vytváranie nových databáz, prípadne tabuliek, importovania dát do databáz, zobrazovania dát v podobe tabuliek a podobne. Pre tieto účely využívam aplikácie phpMyAdmin, pomocou ktorého pracujem s databázou, ako spomínané vytváranie databáze, tabuľky, vkladanie dát alebo import nových dát. Tieto jednoduché operácie by boli možné do implementovať v rámci rozšírenia aplikácie.

Ďalej aplikácia počíta s istou formou uloženia trénovacích dát v tabuľke. Pri procese klasifikácie musí byť v prvom stĺpci tabuľky hodnota triedy do ktorej ja daná vzorka klasifikovaná. Túto skutočnosť nepovažujem za obmedzenie, ale databázová tabuľka musí spĺňať túto požiadavku v rámci mojej implementácie. Ako bolo spomenuté na začiatku, dáta prechádzajú mnohými úpravami než príde k samotnému dolovaniu a použitiu nejakej dolovacej úlohy. V mojej aplikácii sa práve vyžaduje mať trénovacie dáta v spomenutej podobe.

Pri použití skutočne veľkého množstva dát, aplikácia spotrebuje veľa operačnej pamäte kvôli spôsobu, akým ukladám výsledné hodnoty do premenných. Použitie dátového typu `String`, ako indexu resp. kľúča, vzniká väčšia pamäťová náročnosť. Tá rastie exponenciálne v závislosti na počte stĺpcov tabuľky a počte rôznych hodnôt v jednotlivých stĺpcoch, keďže aplikovaná metóda počíta jednotlivé pravdepodobnosti pre všetky kombinácie hodnôt v jednotlivých atribútoch. V praxi sa určite uplatňujú optimalizácie, aby sa pamäťová náročnosť zmestila do prijateľných hodnôt. Možným riešením by bolo napríklad vyriešiť spôsob indexovania na základe čísiel namiesto reťazcov. To by však spôsobilo problém pri zadávaní nových dát užívateľom, kde by sa musel riešiť problém akým spôsobom budem reťazce znakov, ktoré užívateľ zadá prevádzať na číselné hodnoty. S tými by som potom mohol vytvárať pamäťovo menej náročné kľúče resp. indexy pomocou ktorých sa výsledné hodnoty v kolekcii ukladajú. Tento problém by mohol byť vyriešený v rámci rozšírenia a pokračovania na projekte, v prípade nasadenia aplikácie na reálne vzorky dát.

Doteraz som nemal možnosť vidieť ani som sa nestretol s inými reálnymi dátami, ktoré sú vstupom pre klasifikačný model, okrem známej databáze húb *agricius-lepiota*, ktorú som používal pri testovaní aplikácie a modelu. Tento dátový súbor (anglicky dataset) mi poskytol možnosť pracovať s reálnymi dátami. No pri experimentovaní s aplikáciou som zistil, že množina je príliš veľká.

7 Testovanie

Po implementácii nastupuje veľmi dôležitá časť, ktorá tvorí životný cyklus programu. Vynechanie tejto fázy by malo za následky nepredvídané chovanie aplikácie a prípadne jej chybnú činnosť. Seba lepšie navrhnutý program je bez overenia jeho funkčnosti prakticky nepoužiteľný. Testovanie však zaberá veľkú časť z celkovej práce na programe. Úlohou tejto fáze je odhalenie a následná oprava chýb.

Aplikácia bola testovaná počas celého vývoju a implementácie. Každá pridaná funkcionalita, metóda bola samostatne otestovaná a overená jej správnosť aby sa zabránilo nesprávnej funkčnosti hneď s počiatku. Následne sa pridávala ďalšia časť, ktorá sa testovala samostatne a potom spolu s už overenou funkcionalitou, pokiaľ sa celý proces implementácie neukončil. Potom nasledovalo overenie logickej funkčnosti použitých postupov a algoritmov. Tieto testy boli realizované na mojich vlastných dátach a zároveň aj na vzorku reálnych dát.

7.1 Vyhodnotenie úspešnosti modelu

Aby sme zistili nakoľko náš klasifikačný model klasifikuje správne nové dáta, bolo dôležitou časťou testovanie samotného klasifikačného modelu. Proces prebiehal zadávaním dát u ktorých som vedel do akej triedy majú byť klasifikované. Dáta sa vyberali z testovacej množiny a po absolvovaní procesu som porovnal výsledky implementovaného modelu a skutočné, pravdivé výsledky. Výsledky klasifikácie sú v nasledujúcej tabuľke. Ako testovacia množina mi poslúžila databáza už spomínaných húb. Posledný atribút za pomlčkou je klasifikačná trieda, v tomto prípade sa klasifikovalo do dvoch tried. Zisťovali sme či je daná huba jedlá alebo otravná. Znak *p* značí triedu otravných húb, znak *e* triedu jedlých. Z číselných hodnôt je zrejmé, do ktorej triedy sa budú dáta klasifikovať, v tomto prípade sa však jedná o zistenie úspešnosti klasifikačného modelu. Aby som toho dosiahol, náhodne som vyberal dáta z testovacej množiny a jednotlivo porovnával výsledky môjho klasifikačného modelu a zaradenie do tried podľa testovacej množiny.

Výsledky klasifikácie	Testovacia množina	Vyhodnotenie
x-s-w-f-n-f-w-b-p-p: 6.5040986E-5% x-s-w-f-n-f-w-b-p-e : 0.018258657%	x-s-w-f-n-f-w-b-p-e	správne
f-f-n-t-n-f-c-b-n-p : 4.820136E-5% f-f-n-t-n-f-c-b-n-e : 0.49208164%	f-f-n-t-n-f-c-b-n-e	správne
x-s-w-t-p-f-c-n-n-p : 0.037975486% x-s-w-t-p-f-c-n-n-e : 0.014127305%	x-s-w-t-p-f-c-n-n-p	správne
f-f-w-f-n-f-w-b-n-p : 1.8520723E-5% f-f-w-f-n-f-w-b-n-e : 0.034777652%	f-f-w-f-n-f-w-b-n-e	správne
f-y-w-t-p-f-c-n-k-p : 0.009048274% f-y-w-t-p-f-c-n-k-e : 0.008648341%	f-y-w-t-p-f-c-n-k-p	správne
x-s-y-t-a-f-c-b-k-p : 0.028005539% x-s-y-t-a-f-c-b-k-e : 0.007891753%	x-s-y-t-a-f-c-b-k-e	chybne
b-s-w-t-l-f-c-b-n-p : 1.6331016E-4%	b-s-w-t-l-f-c-b-n-e	správne

b-s-w-t-l-f-c-b-n-e : 0.004204041%		
x-y-w-t-p-f-c-n-n-p : 0.037975486% x-y-w-t-p-f-c-n-n-e : 0.02638603%	x-y-w-t-p-f-c-n-n-p	správne
x-s-g-f-n-f-w-b-k-p : 2.7885048E-5% x-s-g-f-n-f-w-b-k-e : 0.01321508%	x-s-g-f-n-f-w-b-k-e	správne
x-y-y-t-a-f-c-b-n-p : 0.047696933% x-y-y-t-a-f-c-b-n-e : 0.036806367%	x-y-y-t-a-f-c-b-n-e	chybne
b-s-w-t-a-f-c-b-g-p : 2.232405E-4% b-s-w-t-a-f-c-b-g-e : 7.4390485E-4%	b-s-w-t-a-f-c-b-g-e	správne
b-s-y-t-a-f-c-b-g-p : 1.1811666E-4% b-s-y-t-a-f-c-b-g-e : 5.313607E-4%	b-s-y-t-a-f-c-b-g-e	správne
x-y-w-t-p-f-c-n-p-p : 0.08117696% x-y-w-t-p-f-c-n-p-e : 0.022577105%	x-y-w-t-p-f-c-n-p-p	správne
f-s-n-f-n-f-w-b-k-p : 4.948209E-6% f-s-n-f-n-f-w-b-k-e : 0.012039887%	f-s-n-f-n-f-w-b-k-e	správne
f-s-n-t-p-f-c-n-p-p : 0.022483794% f-s-n-t-p-f-c-n-p-e : 0.017030839%	f-s-n-t-p-f-c-n-p-p	správne

Z tabuľky je možné vypočítať percentuálnu pravdepodobnosť úspešnosti správnej klasifikácie modelom. Pre úplné zhodnotenie by bolo potreba previesť kontrolu najlepšie s celou testovacou množinou a následne previesť podobné porovnanie ako je znázornené v tabuľke. Z dát v tabuľke vychádza, že model z pätnástich prípadov klasifikoval dva krát chybne, čo znamená úspešnosť modelu z približne 86,67% po zaokrúhlení na dve desatinné miesta. Nemôžeme však tvrdiť, že táto hodnota platí pre celý klasifikačný model, bolo by treba previesť klasifikáciu viacerých záznamov tabuľky z trénovacej množiny a následných porovnaní výsledkov.

8 Záver

Cieľom tejto bakalárskej práce bolo navrhnúť aplikáciu a implementovať vybranú klasifikačnú metódu (naivná Bayesovská klasifikácia). Ďalej overiť jej funkčnosť na vhodne zvolenej vzorke dát a nakoniec previesť experimenty s implementovaným modelom, zhodnotiť dosiahnuté výsledky a prípadné rozšírenia projektu.

K dosiahnutiu tohto cieľu bolo nutné preštudovať problematiky získavania znalostí z databáz a ďalej sa presnejšie zamerať na klasifikáciu ako jeden typ úloh pri dolovaní z dát. V tejto časti som sa zoznámil s rôznymi druhmi metód a ich princípmi. Pred samotnou implementáciu bolo treba previesť návrh celej aplikácie s dôrazom na aplikáciu vybranej klasifikačnej metódy (naivná Bayesovská klasifikácia). Navrhol som užívateľské rozhranie v ktorom sa dajú pohodlne zadávať nové vstupné dáta u ktorých chceme previesť klasifikáciu a následne vhodne zobraziť výsledky. Dôležitou časťou práce bolo zhodnotenie výsledkov klasifikačného modelu, aby sa dalo určiť ako bude úspešný pri klasifikovaní nových dát. Za týmto účelom som prevádzal sériu experimentov na mojich vlastných dátach, ale hlavne na reálnom vzorku dát a porovnával výsledky s testovacou množinou. Dospel som k záveru, že klasifikačný model funguje správne a podáva očakávané výsledky.

Budúce rozšírenie aplikácie by som videl v prípadnom pridaní nových klasifikačných metód do aplikácie alebo rozšírenie aplikácie o základnú prácu s databázou, ako vytváranie databáz, tabuliek, importu dát do tabuliek a zobrazenie tabuliek. Všeobecne by sa dalo na aplikácii pokračovať pridaní vlastností a nedostatkov spomenutých v podkapitole 6.6.

Literatúra

[1] Zendulka, J., Bartík, V., Lukáš, R., Rudolfová, I.: Získávání znalostí z databází. Studijní opora, 2006, Fakulta informačních technologií VUT v Brně.

[2] Lukáš, R.: Klasifikace a predikce [prezentace]. 2008 [cit. 2010-05-16] Dostupný z WWW: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ZZN-IT/lectures/2008/05_klasifikace_predikce.ppt>.

[3] Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Second Edition. Morgan Kaufmann Publishers, 2006. ISBN 13: 978-1-55860-901-3, ISBN 10: 1-55860-901-6.

Zoznam príloh

Príloha 1. CD so zdrojovými súbormi, spustiteľnou aplikáciou a súborom obsahujúcim vzorku dát.